Part 1: Stack ADT (15 Points)

Write a C++ program that reads a text file that contains an encrypted letter written in English. The letter is encoded in a way that all lines are sorted in a reversed order. That is, the first line is stored last, and the last line is stored first. Moreover, the words in each line are reversed (in a particular way demonstrated in the figure below) as well. For each line, letters in the words with an odd rank (position) are reversed. That is, the first word, the third word, and the fifth word, etc, is reversed. All words with an even rank are kept with no change (in their original order and not reversed).

The following example demonstrates the way messages are encrypted:

Original Message:

Line1	Let us meet at Appleton downtown and get some ice cream.
Line2	I will pay for the ice cream.
Line3	Please bring your books with you.

The encoded message as stored in the input text file:

Line1	esaelP bring ruoy books htiw you
Line2	I will yap for eht ice maerc
Line3	teL us teem at notelppA downtown dna get emos ice maerc

Hints:

-Think of a data structure we studied in class that might be a good fit for this program. Yes, it is the **Stack**. You are expected to (should) use Stack ADT as a data structure to write this program. Use one for the lines and one for the words.

-You can use the STL stack library <stack> or any of the implementations I did share with you in class (find them on Canvas). Though, I recommend that you use the Linked List-based implementation or the dynamic-array-based. Or, you can use the STL Staack class(#include <stack>).

-If you are not using the STL stack, have the stack implementation stored in a header file and include it in your project if you want to use our implementation, not the STL version.

Part 2: Practicing Doubly Linked List ADT (30 Points)

Problem Description:

In this project, you are expected to design and fully implement a car rental system (a subsystem of a full system) that controls the vehicle's partial activities within the inventory. The system is expected to help the inventory automate its vehicle reservation and query processes (without getting involved in the customer information). It uses a data structure to keep track of the vehicle inventory in the system (please use Doubly Linked List). Moreover, vehicle information

is already provided in a text file (vehicles.txt Download vehicles.txt), and no need to worry about inserting new vehicles into the file. However, the system needs to update the file after each reservation or return process.

Your system should be able to perform at least the following operations:

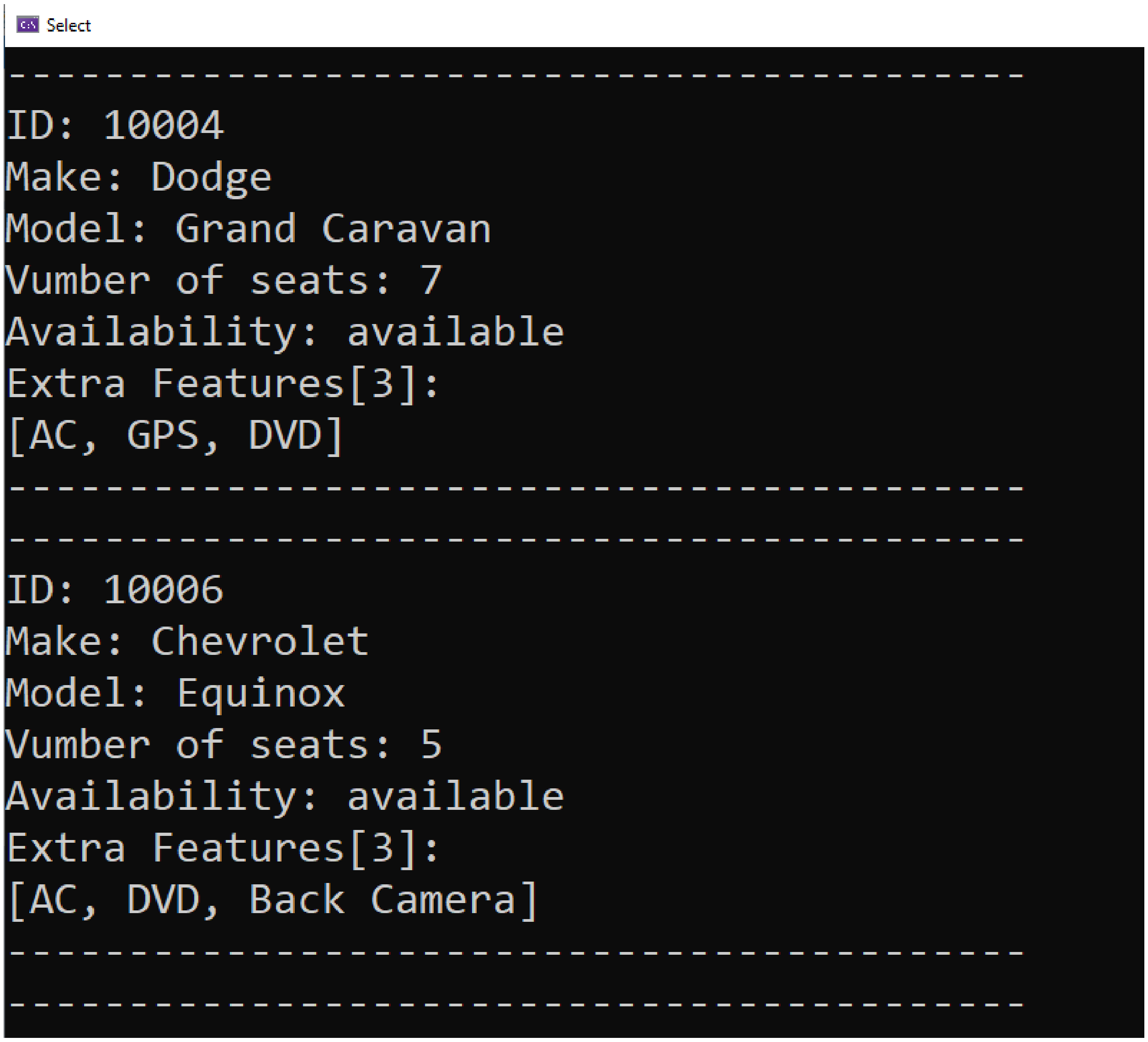
1. Print All of the registered vehicles. (available and already reserved)
2. Print list of the available vehicles.
3. List of the reserved vehicles (rented currently by customers).
4. Check availability.
5. Reserve a vehicle.
6. return a reserved vehicle.
7. Terminating the program when needed.
8. Exit

Note: If you add the insertion option, you will get extra points.

Clearing the console screen can be done using: **system("cls");**

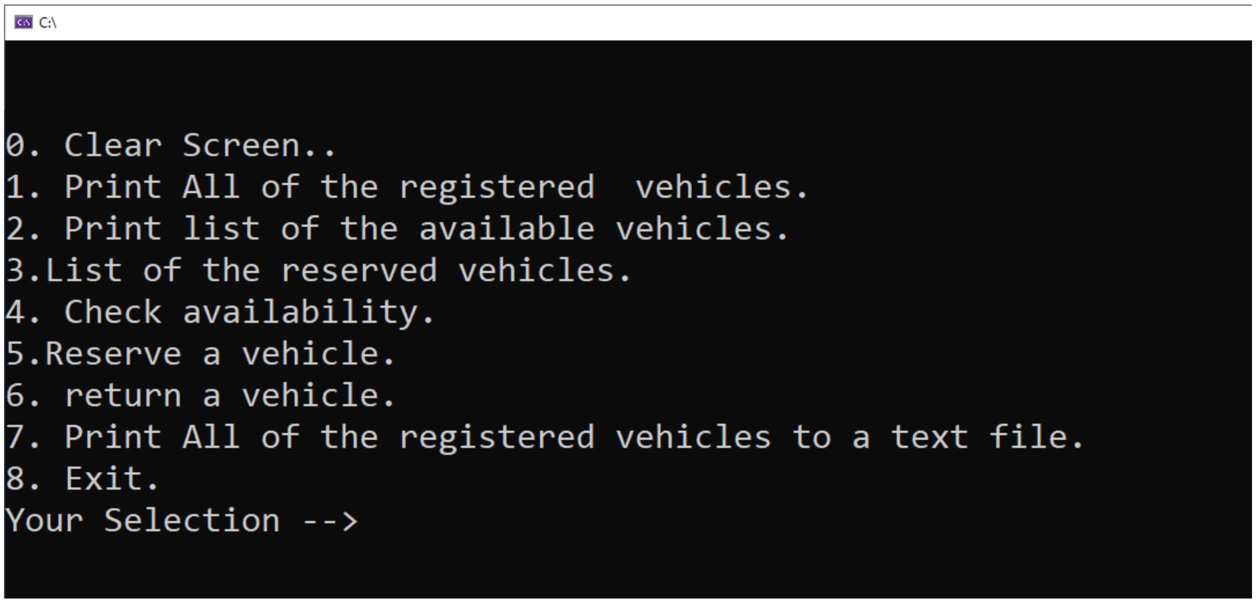
Example Screens:

This is how records are printed:



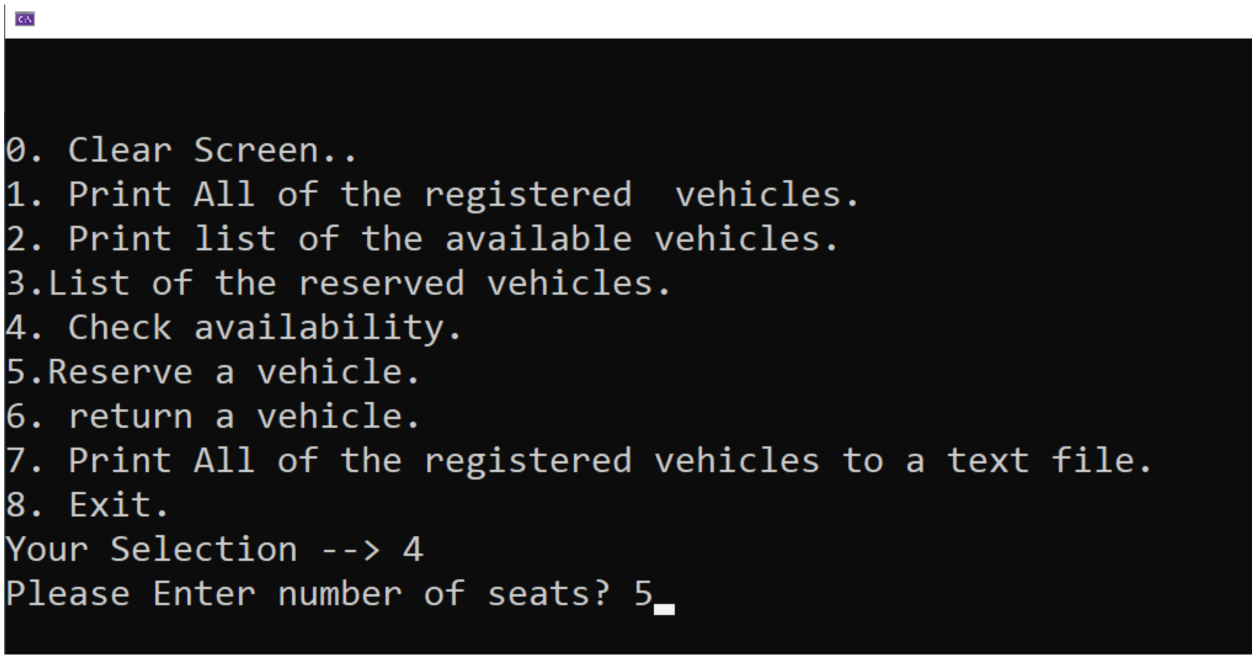
```
------------------------------------------
ID: 10004
Make: Dodge
Model: Grand Caravan
Vumber of seats: 7
Availability: available
Extra Features[3]:
[AC, GPS, DVD]
------------------------------------------
------------------------------------------
ID: 10006
Make: Chevrolet
Model: Equinox
Vumber of seats: 5
Availability: available
Extra Features[3]:
[AC, DVD, Back Camera]
------------------------------------------
------------------------------------------
```

Main Menue:



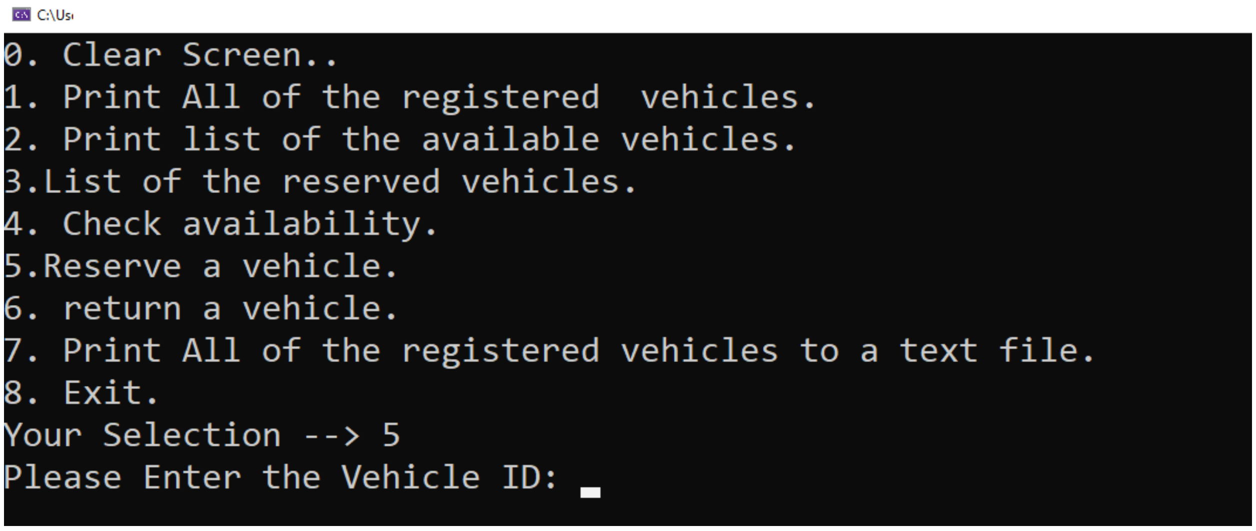
```
0. Clear Screen..
1. Print All of the registered  vehicles.
2. Print list of the available vehicles.
3.List of the reserved vehicles.
4. Check availability.
5.Reserve a vehicle.
6. return a vehicle.
7. Print All of the registered vehicles to a text file.
8. Exit.
Your Selection -->
```

Check Availability:



```
0. Clear Screen..
1. Print All of the registered  vehicles.
2. Print list of the available vehicles.
3.List of the reserved vehicles.
4. Check availability.
5.Reserve a vehicle.
6. return a vehicle.
7. Print All of the registered vehicles to a text file.
8. Exit.
Your Selection --> 4
Please Enter number of seats? 5_
```

Reserve A Vehicle:


C:\Us

```
0. Clear Screen..
1. Print All of the registered  vehicles.
2. Print list of the available vehicles.
3.List of the reserved vehicles.
4. Check availability.
5.Reserve a vehicle.
6. return a vehicle.
7. Print All of the registered vehicles to a text file.
8. Exit.
Your Selection --> 5
Please Enter the Vehicle ID: _
```

Part 3: Practicing Rule of Three: (15 Points)

Please implement the rule of three for the Stack ADT and Queue ADT studied in class.

Stack ADT:

```
#include <iostream>
using namespace std;
template <class T>
class Stack {
public:
 Stack() {
  top = -1;
  data= new T[CAPACITY];
 }
 ~Stack() {
  delete[] data;
 }
 void push(T item);
 void pop();
 bool empty() { return (top == -1); }
 int size() { return (top + 1); }
 int getCAPACITY() { return CAPACITY; }

 T atTop();
private:
 int CAPACITY = 10;
 int top;
 T* data;
 bool expand(int newSize); //expanding the array
 bool shrink(int newSize);//reducing the size of the array
};
template <class T>
void Stack<T>::push(T item) {
 //pre-conditions:  stack is not Full.
 if (top < (CAPACITY - 1)) {
  ++top;
  data[top] = item;
 }
 else
 {
  //cout << "Stack Overflow!\n";
  int newCAPACITY = CAPACITY + (CAPACITY / 4);
  if (expand(newCAPACITY)) {
   ++top;
   data[top] = item;
  }
  else
   cout << "Problem while expanding the stack!\nunsuccessful push
operation.";
 }
}
template <class T>
void Stack<T>::pop() {
 //pre-conditions:  stack is not empty

 if (!empty()) {
  --top;
  if((top+1)<= (CAPACITY/2) && (top + 1)>10)
  {
   int newCAPACITY = CAPACITY - (CAPACITY / 4);
    if (!shrink(newCAPACITY))
     cout << "Problem while shrinking the stack!\n";
  }
 }
 else
  cout << "Stack Underflow!\n";
}

template <class T>
T Stack<T>::atTop() {
 //pre-conditions:  stack is not empty
 if (!empty())
  return data[top];
 else {
  cout << "Stack Underflow!\n";
  exit(1); //terminate the program
 }
}
template <class T>
bool Stack<T>::expand(int newCAPACITY) { //expanding the array
 if (newCAPACITY > CAPACITY) {
  T* tmpArray = new T[newCAPACITY];
  for (int i = 0; i < size(); ++i)
   tmpArray[i] = data[i];
  delete[] data;
  data = tmpArray;
  CAPACITY = newCAPACITY;
  return true;
 }
 else
```

```
  return false;
}
template <class T>
bool Stack<T>::shrink(int newCAPACITY) {//reducing the size of the array
 if (newCAPACITY < CAPACITY) {
  T* tmpArray = new T[newCAPACITY];
  for (int i = 0; i < newCAPACITY; ++i)
   tmpArray[i] = data[i];
  delete[] data;
  data = tmpArray;
  CAPACITY = newCAPACITY;
  return true;
 }
 else
  return false;
}
```

Queue ADT:

```
// ArrayBasedQueuePrj.cpp : This file contains the 'main' function. Program
execution begins and ends there.
//

#include <iostream>
#include <string>
using namespace std;

template <class T>
class ArrayQueue {
public:

 ArrayQueue();
 ArrayQueue(int);
 void enqueue(T item);/*Inserts an element at the rear of the queue. */
 void dequeue();/*Removes the first element of the queue.*/
 T front();/*Returns, but does not remove, the first element of the queue.*/
 bool empty() {
  return (currentNumberOfElements == 0);
 }
 int size() {
  return (currentNumberOfElements);
 }
 bool expand(int newSize); //expanding the dynamic array
 void printQueue();
 template <class T> friend  ostream& operator<<(ostream& os, const
ArrayQueue<T>&);
/*Default array capacity. */
  int CAPACITY = 10;
private:

 int first; // index of the front element
 int last; // index of the last element
 int currentNumberOfElements = 0;// current number of elements
/** Generic array used for storage of queue elements. */
```

```
 T* data; // generic array used for storage

};
template <class T>
ArrayQueue<T>::ArrayQueue() {    // constructs queue with default capacity
 data = new T[CAPACITY];
 first = 0;
 last = -1;
 currentNumberOfElements = 0;
}

template <class T>
ArrayQueue<T>::ArrayQueue(int CAPACITY_) {  // constructs queue with given
capacity
 CAPACITY = CAPACITY_;
 data = new T[CAPACITY];
 first = 0;
 last = -1;
 currentNumberOfElements = 0;
}

template <class T>
void ArrayQueue<T>::enqueue(T e) {

 if (currentNumberOfElements < (CAPACITY )) {
  int avail = (last + 1) % CAPACITY;   // use modular arithmetic
  data[avail] = e;
  ++currentNumberOfElements;
  last = (last + 1) % CAPACITY;
 }
 else
 {

  int newCAPACITY = CAPACITY + (CAPACITY / 4);
  if (expand(newCAPACITY)) {
   int avail = (last + 1) % CAPACITY;   // use modular arithmetic
   data[avail] = e;
   ++currentNumberOfElements;
   last = (last + 1) % CAPACITY;
   cout << "\nexpansion. New Capacity is"<< CAPACITY <<endl;
  }
  else
   cout << "Problem while expanding the queue!\nunsuccessful push
operation.";
 }

}

template <class T>
T ArrayQueue<T>::front() {
 if (empty()) {
  cout << "Queue is empty!\n"; //I would throw an exception.
  return;
 }
 return data[first];
}
```

```
template <class T>
void ArrayQueue<T>::dequeue() {
 if (empty()) {
  cout << "Queue is empty!\n";
  return;
 }
 T answer = data[first];
 first = (first + 1) % CAPACITY;
 --currentNumberOfElements;
}
template <class T>
bool ArrayQueue<T>::expand(int newCAPACITY) { //expanding the array
 if (newCAPACITY > CAPACITY) {
  T* tmpArray = new T[newCAPACITY];
  int j = 0;
  for (int i = first ; i !=last; i = (i + 1) % CAPACITY)
   tmpArray[j++] = data[i];

  tmpArray[j] = data[last];
  delete[] data;
  data = tmpArray;
  CAPACITY = newCAPACITY;
  first = 0;
  last = size() - 1;
  return true;
 }
 else
  return false;
}

template <class T>
void ArrayQueue<T>::printQueue() {

  cout << "[";
  if (!empty()) {
   cout << data[first];
   for (int i=(first+1) % CAPACITY; i!= ((last+1 )%CAPACITY);
i=(i+1)%CAPACITY)
   {
    cout << ", " << data[i];
   }//end for

  }
  cout << "]";

}
template <class T>
ostream& operator<<(ostream& os, const ArrayQueue<T>& queueObj) {

 os << "[";
 if (1) {
  os << queueObj.data[queueObj.first];
  for (int i = (queueObj.first + 1) % queueObj.CAPACITY; i != ((queueObj.last
+ 1) % queueObj.CAPACITY); i = (i + 1) % queueObj.CAPACITY)
  {
   os << ", " << queueObj.data[i];
  }//end for
 }
 os << "]";
 return os;
}
```